TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics and Interdisciplinary Studies

PAUL SABATIER UNIVERSITY TOULOUSE III
IUP of Intelligent Systems

Study program: N2612 – Electrical engineering and informatics

Study field: 2612T071 – Engineering of Interactive Systems

Visualization of Domain Structure in Ferroelectric Materials

Vizualizace doménové struktury ve feroelektrikách

Diploma thesis

Author: **Bc. Jan Pokorný**

Supervisor: Ing. Pavel Márton, Ph.D.

In Liberec 18.5.2012

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL a UPS má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL a UPS, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Acknowledgements

I would like to thank my supervisor Ing. Pavel Márton, Ph.D., for his time and valuable advices during the development an application. Also I would like to thank my family, especially my parents, for their support during the whole international study program.

Abstrakt

Diplomová práce se zabývá vizualizací doménových struktur v aplikačně významné třídě feroelektrických materiálů. Hlavním cílem je úprava stávajících a implementace nových algoritmů do vizualizačního programu `fview`. První důležitou částí je výběr vhodného vizualizačního nástroje, a následné využití jeho funkčnosti k zobrazení doménové struktury. Další část je věnována tvorbě palet používaných pro reprezentaci feroelektrických a feroelastických spontánních stavů. Následuje implementace algoritmu pro vyhledávání souvislých oblastí a vykreslování šipek ve směru feroelektrické polarizace pro každou doménu zvlášť. Poslední část se zabývá vyhledáváním rozhraní mezi doménami. Program umožňuje vizualizaci v několika režimech, které jsou implementovány jako vzájemně se doplňující vrstvy. Parametry jsou programu předávány jako řádkové parametry. V textu se detailně zabývám funkčností mnou navržených řešení a upozorňuji na možné nedostatky popřípadě možné směry dalšího vývoje.

Klíčová slova: Doménové struktury, vizualizace, programování C, feroelektrické materiály

Abstract

This diploma thesis deals with the visualization of domain structures in significant class of ferroelectric materials. The goal is to modify existing and implement new algorithms into the visualization program `fview`. The first important part is the selection of an appropriate visualization tool and subsequently using its functionality to display domain structures. The next part is devoted to creation of palettes used for representation ferroelectric and ferroelastic spontaneous states. The next section solves implementation of algorithm for searching connected areas and drawing arrows in the direction of ferroelectrical polarization for each domain. The last section deals with searching for a sharp interface between domains. The program allows visualization in several modes, which are implemented as complementary layers. Parameters are passed to program as command parameters. In this work I deal with functionality of my solutions and point out to possible weaknesses or possible future development.

Keywords: Domain structures, visualization, programming in C, ferroelectric materials

Abstract

Ce projet de fin d'études porte sur la visualisation des structures de domaines de materiaux ferroélectriques en classes signifiantes. Le but est de modifier des algorithmes existant et d'en implémenter des nouveaux dans le programme `fview`. La première partie la plus importante est la sélection d'un outil de visualisation approprié pour ensuite utiliser ses fonctionnalités pour afficher les structures de domaine. La partie suivante est dévouée à la création de palettes utilisées pour la représentation ferroélectrique et ferroélastiques des états spontanés. La section suivante résout l'implémentation d'un algorithme de recherche de zones connectées et de dessin de flêches dans la direction de la polarisation ferroélectrique pour chaque domaine. La derniere section traite de la recherche d'une forme d'interface entre les domaines. Ce programme permet la visualisation dans différents modes, implémentés en tant que calques complémentaires. Les paramètres sont passés au programme via la ligne de commande. Dans ce travail, je traite des fonctionnalités de mes solutions et souligne leurs possibles faiblesses ou leurs possibles futures améliorations.

Mots clés: la structure des champs, la visualisation, programmation C, les matériaux ferroélectriques

Content

List of figures

Introduction

Ferroelectric materials are subclass of dielectric materials, important for applications due to their excellent piezoelectric and dielectric response. The ferroelectric properties were observed and described for the first time in 1921 by Joseph Valasek with Rochelle salt. The most widely known ferroelectric material is perhaps $BaTiO_3$. Ferroelectrics possess spontaneous polarization, which can be switched by external electrical field or mechanical loading. Each ferroelectric phase is basically limited to given temperature interval. When the temperature increases and reaches the so-called Curier point, a transition to paraelectric state begins and the ferroelectric material loses its properties and spontaneous polarization disappears. Close to Curier point the dielectric and piezoelectric properties are significantly enhanced. In ferroelectric phase, there exist regions with the same spontaneous polarization, called domains. Domain structure can change by mechanical action and by electrical field. Directions of spontaneous polarization are related to crystallographic orientations of the ferroelectric crystal and there are only few of them (6 in tetragonal, 8 in rhombohedral and 12 in orthorhombic phase) [1, 2].

Ferroelectrics are used in actuators, oscillators, microphones, sensors, micro-electro-mechanical devices (MEMS), capacitors, or even for construction of faster and non-volatile memory (FRAM), and other novel applications are proposed.

The diploma thesis deals with the visualization of the domain structures in ferroelectric materials using a program called `fview` (visualization of data from simulations). It is program written in C/C++ language [3] using a freeware third-party visualization library. The first chapter is devoted to research of the right tools/libraries for visualization. Because `fview` already existed [4] and used to visualize simulation outputs, next part deals with description of program `fview` and differences between the previous and new version. More specific options of program are also described. Inseparable part is description about palettes construction technique. Next chapter deals with implementation flood fill algorithm, different types, pros and cons. The main reason for flood fill is ability to draw one big arrow inside every domain. Last chapter solves problem of finding sharp interface among domains, which is actually done by interpolation using Ferguson's cubic.

1 Research

1.1 Interface for visualization

In this first chapter I deal with choosing the right visualization tool. The tool can be stand alone program or library. Library has obvious advantage that is easier to use and as a general rule is smaller. Short survey of thirteen visualization tools follows. Important requirements are:

- Usable under Linux operating system as a freeware. The better option is cross-platform (Windows, Linux and OS X).
- Ability to draw 2D maps good enough.
- Support controlling via script (non-interactive control).
- Support various output image formats.
- Support and supposed future support of the tool.
- Good manual.

During my research I found tools, which are not suitable because they do not fulfill one or more of the main requirements. **RLPlot** [5] is a tool which doesn't support scripting at all; it can be controlled only interactively. Next problem of found tools were their age (latest usable version) and any information about new version. These tools are **OpenDX** [6], **SciDAVis** [7] and **Ploticus** [8]. **Easyviz** [9] is special kind of tool defined as unified interface for visualization. It uses other visualization programs to visualize data using Python scripts, so it cannot visualize data by itself. Other tools seem to be good so I will describe them in more detail and try to choose the best one.

QtiPlot [10] offers basic functions for creating graphs in 2D or 3D. This program is free only under Linux systems. It supports export to various image (bitmap or vector) formats. The last stable version was released in November 2011 and there is no information about new version, but the last version is still supported and with high probability will be supported to the future. QtiPlot is scriptable via Python.

GLE (Graphics Layout Engine) [11] is a graphical scripting language written in C++. Complex pictures can be drawn with user defined subroutines and scripts. Plots can be 2D or 3D. Last release was developed in March 2012. For support bitmap output, additional libraries have to be installed.

PLplot [12] is a library written in C. For our purpose library has advantage that functionality can be easily implemented without scripting and program is easily portable. It has a wide range of plot types including 2D and 3D graphs. PLplot version 5.9.9 was released in October 2011 and next version 5.10.0 is reported.

Matplotlib [13] is a pure Python plotting library. Syntax is familiar to MATLAB and it serves as free replace for MATLAB users. In basic version offers 2D plotting. For 3D plotting it needs to be next add-on toolkit. Matplolib is dependent on other library named NumPy and installation is not so easy.

Gnuplot [14] is well-known widespread command-line driven graphing utility. It supports plots in 2D and 3D with many different types of output. Gnuplot can be controlled non-interactively and it is also used as a plotting engine by third-party applications like Octave. Gnuplot has very big community and support. Last version 4.6.0 was released in March 2012

GNU Octave [15] is a high-level language quite similar to MATLAB, which uses Gnuplot as its backend. It means anything that can be plotted with Gnuplot can be plotted with GNU Octave. It supports scripting and interactive mode. GNU Octave offers a lot of functions, for our purpose lot of useless functions therefore the size is big. Octave does have a richer language for computation, which has its obvious advantages, but it's still limited by Gnuplot.

Scilab [16] is similar to GNU Octave. It provides more than 1 500 mathematical functions, 2D or 3D graphs can be drawn. Scripting is very similar to Octave or MATLAB. Scilab has more useless add-ons and bigger hardware requirements than Octave. Installation takes about 250MB.

MayaVi2 [17] is mainly focused on visualization of 3D data. It requires four libraries to be installed before MayaVi2. This program can be used as graphical application or as a plotting engine from Python scripts. As all other programs is cross-platform and free.

1.2 PLplot library

PLplot is a cross-platform library for creating many types of graphs and plots (x-y plots, semi-log plots, log-log plots, contour plots, 3D surface plots, mesh plots, bar charts and pie charts). PLplot is written in C language, but there are many compiled and interpreted other languages such Java, Fortran, Perl, Python and others which has access to PLplot. Plotting can be accomplished in non-interactive and interactive way.

Advantage of PLplot is that has support for Unicode. PLplot supports different file formats for plotting such as CGM, GIF, JPEG, LaTeX, PBM, PDF, PNG, PostScript, SVG and Xfig. This library is free software primarily licensed under the LGPL.

1.3 PLplot library installation

PLplot is built using CMake-based build system. The installation under Linux consists of few steps after downloading. Following commands should be entered to terminal in order to install PLplot library. Variable *yourname* depends on the name which was filled during the Linux installation process. Last variable *$PL_VERSION* should be directly replaced by version of PLplot we want to install or it can be set as global variable.

```
1. cd /home/yourname/plplot
2. rm -rf plplot-$PL_VERSION build_dir install_directory
3. tar -zxf plplot-$PL_VERSION.tar.gz
4. mkdir build_directory
5. cd build_directory
6. cmake -
   DCMAKE_INSTALL_PREFIX=/home/yourname/plplot/install_directory
   \plplot-$PL_VERSION/ >& cmake.out
7. less cmake.out
8. make VERBOSE=1 >& make.out
9. less make.out
10. make VERBOSE=1 install >& make_install.out
11. less make_install.out
```

From PLplot 5.9.3 the standard output devices as png, jpg or gif were deprecated. PLplot authors recommend using cairo-based device which provides a vector graphic-based output and is also designed to use hardware acceleration when available. But there is still possibility how to use deprecated output formats in PLplot 5.9.9. It is

necessary to compile it with another option as follows (for png format in this case).

```
cmake -
DCMAKE_INSTALL_PREFIX:PATH=/home/yourname/plplot/install_directory
\plplot-$PL_VERSION -DPLD_png=ON >& cmake.out
```

2 Program fview

`Fview` was originally developed in C/C++ using PGPLOT library for visualization of domain structures. Last stable version of PGPLOT library was developed in 2001 and from the same year library is no longer supported. `Fview` was completely remade using PLplot library and new functions were added. Detailed description of changes will be described in following chapters.

`Fview` consists of files:

- display.cpp – main visualization routines
- fview.cpp – loading of parameters and array
- utility.cpp – additional functions
- plplot.h – library providing visualization functions

2.1 Input parameters

`Fview` is primarily designed for non interactive control. It means it can take input parameters from command line. This advantage is used when user wants to process many files at once (generally written in script) without any additional interaction. Following table shows every possible command.

Switch	Meaning
-h	Shows help.
-i	Specifies path to input file. If no input file is specified, chosen one or all palettes are printed.
-t	Specifies path to file with defects.
-o	Specifies name of output file (if not, name and path of input file is used).
-d	Set output device (gif, ps, png). Default value is png.
-p	Set palette.
-m	Choose dimension of image (2D or 3D). Default is 2D.
-l	Layers – color, small arrows, big arrows, borders, sharp interface.
-s	Size to scale image resolution.
-v	Verbose output (shows chosen parameters).

More detailed overview contains following help directly printed by `fview`.

```
fview [-i filename] [-t defectfile] [-o outfilename] [-h] [-d device] [-p palette] [-m
dimension] [-l layers] [-s size] [-v]

 +----------------+----+-------------------------------------------------------------+
 | --input_file   | -i | Input file                                                  |
 | No input file  |    | Print chosen palette or all palettes                        |
 | --defect_file  | -t | Input file with defects                                     |
 | --output_file  | -o | Name of output file                                         |
 | --help         | -h | Print this help                                             |
 | --device       | -d | Output device: gif, ps, png (default)                       |
 | --palette      | -p | Number of palette to use:                                   |
 |                |    |    0 - Tetragonal + Orthorombic + Rhombohedral              |
 |                |    |    1 - Tetragonal                                           |
 |                |    |    2 - Orthorombic                                          |
 |                |    |    3 - Rhombohedral                                         |
 |                |    |    4 - HSL x,y palette                                      |
 |                |    |    5 - Tetragonal without difference between opposite vectors |
 |                |    |    6 - HSL Color model                                      |
 |                |    |    9 - Tetragonal grayscale                                 |
 | --dimension    | -m | Dimension of output: 2-all images 2D, 3-3D view of 3D images  |
 | --layers       | -l | color (1) | small_arrows (2) | big_arrows (4) | borders (8)   |
 |                |    | sharp_interface (16)                                        |
 | --size         | -s | Coefficient to scale image resolution                       |
 |                |    | (width - s*800; height - s*600)                             |
 | --verbose      | -v | List all chosen options                                     |
 +----------------+----+-------------------------------------------------------------+
```

As we can see each palette is defined by unique number. The most important switch is *–l* which contains five complementary options. They can be entered by words side by side with any separator (for example #) or by number given in brackets (or simply by their sum for more options).

Other important functionalities of visualization are determined by value of following variables, defined at the very top of the file display.cpp.

- `max_g_search` - maximal number of points for searching gradient
- `curve_pts` - number of points which creates curve between 2 points (Ferguson)
- `radius_neighbour` - minimal distance between 2 points in the interface
- `radius_neighbour_search` - radius for searching neighbors
- `radius_endpoints_search` - radius for searching endpoints
- `max_endpoints_number` - maximal number of endpoints which can be found and connected together
- `inter_width` - set width of sharp interface

2.2 Structure of input files

Files which are loaded by `fview` need to have exact structure. Otherwise `fview` will not be able to read them and program will close. First line in the file with data contains four arguments which give necessary information about the structure that follows. First character on the first line is grid then follows first argument, number which determines if the array is 2D or 3D. Second parameter is length of the array (image) in x-axis, next parameter is length in y-axis and the last parameter means how many values in a row is taken for compute polarization, in other words if polarization is one, two or three dimensional. After that line continues data (position of points in space). One example follows.

```
#2      200     240     2
```

This first line means that data are 2-dimensional, width of picture is 200px, and height is 240px with two possible polarizations.

3 Palettes

Palettes are a fundamental element for data visualization. In this chapter I will explain how palettes are created and describe all nine palettes which can be used in `fview`.

3.1 Palettes construction

Principle of palette construction will be demonstrated on one of the basic palettes - tetragonal. Every palette has its own number of spontaneous states. Basically, the number of spontaneous states means how many different full colors palette have.

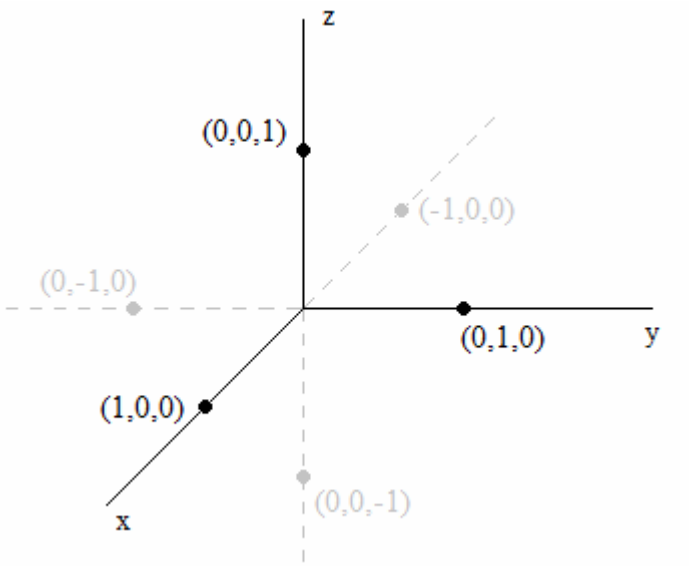


Figure 3.1: Spontaneous states for tetragonal palette in space

Figure 3.1 demonstrates points of tetragonal palette in space. From the picture is obvious that tetragonal palette has 6 spontaneous states. Distance between any point and the origin of coordinates is (independently on palette) always 1, so we can consider it as unity sphere. Points in space should be represented by several ways. We will use classical description by Cartesian coordinates with x, y and z component and spherical coordinates composed of distance r and angles θ and φ. Range of inclination angle φ is from -180° to 180° (horizontal axis in every palette). Elevation angle θ is from -90° to 90° (vertical axis in every palette). Spherical coordinates can be converted to obtain Cartesian coordinates by following equations (1).

$$x = r \cdot \cos\varphi \cdot \cos\theta$$
$$y = r \cdot \sin\varphi \cdot \cos\theta$$
$$z = r \cdot \sin\theta \tag{1}$$

Transformation from Cartesian to spherical coordinates is little bit complicated (2). There is an obvious issue if coordinate x is equal zero. Then for angle φ occurs division by zero.

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = \arcsin\left(\frac{z}{r}\right)$$
$$\varphi = \arctan\left(\frac{y}{x}\right) \tag{2}$$

From this reason it is necessary to redefine computation for angle φ (3). This function is also known as *atan2* in programming languages. The range of output angle is (-π, π]. Table 1 shows spontaneous states for tetragonal palette expressed in both coordinates.

$$\varphi = \begin{cases} \arctan\left(\frac{y}{x}\right) & x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & y \geq 0, x < 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & y < 0, x < 0 \\ \frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ 0 & y = 0, x = 0 \end{cases} \tag{3}$$

Table 1: Tetragonal Cartesian and equivalent spherical coordinates

Spontaneous state	Cartesian coordinates			Spherical coordinates		
	x	y	z	r	θ	φ
1	1	0	0	1	0°	0°
2	-1	0	0	1	0°	180°
3	0	1	0	1	0°	90°
4	0	-1	0	1	0°	-90°
5	0	0	1	1	90°	0°
6	0	0	-1	1	-90°	0°

In the Figure 3.2 is visualized tetragonal palette. We can observe 6 different full colors. If the point in the space is close or is exactly between two spontaneous states we can observe thin white contour among full colors.

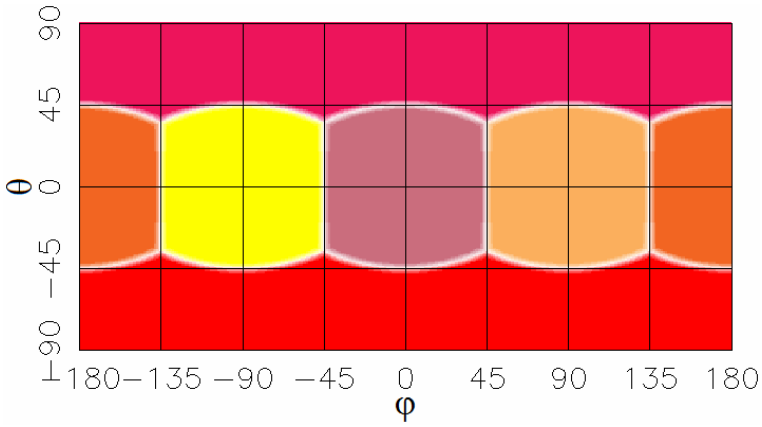


Figure 3.2: Tetragonal palette

Actually it's not just white color. Figure 3.3 shows a method of assigning colors between full colors. S1, S2, S3 and S4 are spontaneous states with its full color. So the contour consists of 8 colors gradually changing from full color to white color. The number of colors is not randomly selected, but computed as follows.

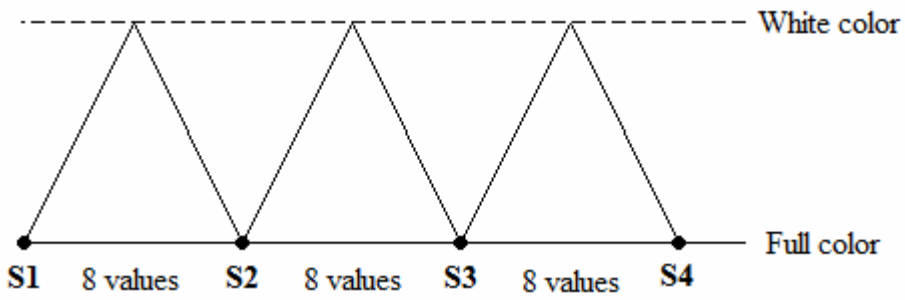


Figure 3.3: Assigning lighter shades of full colors

Number of colors for basic palettes is restricted to 1 byte (256). PLplot has predefined 16 basic colors which can be changed, but I keep them and use only the rest 240 colors. Basic palettes include already mentioned tetragonal palette, next rhombohedral palette with 8 spontaneous states and orthorhombic palette with 12 spontaneous states. The last one is created by mixing these three palettes together, so it gives maximal number of spontaneous states for basic palette is 26.

$$\frac{240}{26} = 9{,}23077 \cong 9 \tag{4}$$

Using (4) we can compute how many colors can be used among spontaneous states (full colors). To keep some reserve, 8 colors are used.

3.2 Basic palettes

Tetragonal palette has been already defined so I mention here the rest of basic palettes which are rhombohedral, orthorhombic and mix of these three palettes. Table 2 shows definition of rhombohedral palette which has 8 spontaneous states and Table 3 shows orthorhombic palette with 12 spontaneous states.

Table 2: Rhombohedral Cartesian and equivalent spherical coordinates

Spontaneous state	Cartesian coordinates			Spherical coordinates		
	x	y	z	r	θ	φ
1	0.577	0.577	0.577	1	35.26°	45°
2	0.577	0.577	-0.577	1	-35.26°	45°
3	0.577	-0.577	0.577	1	35.26°	-45°
4	0.577	-0.577	-0.577	1	-35.26°	-45°
5	-0.577	0.577	0.577	1	35.26°	135°
6	-0.577	0.577	-0.577	1	-35.26°	135°
7	-0.577	-0.577	0.577	1	35.26°	-135°
8	-0.577	-0.577	-0.577	1	-35.26°	-135°

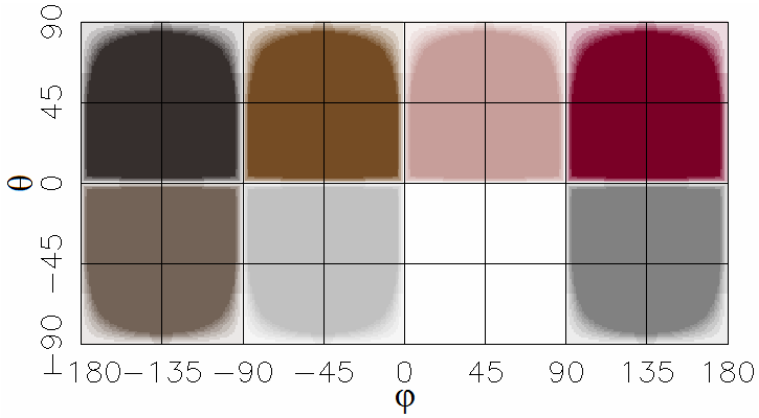


Figure 3.4: Rhombohedral palette

Table 3: Orthorombic Cartesian and equivalent spherical coordinates

Spontaneous state	Cartesian coordinates			Spherical coordinates		
	x	y	z	r	θ	φ
1	0	0.707	0.707	1	45°	90°
2	0	0.707	-0.707	1	-45°	90°
3	0	-0.707	0.707	1	45°	-90°
4	0	-0.707	-0.707	1	-45°	-90°
5	0.707	0	0.707	1	45°	0°
6	0.707	0	-0.707	1	-45°	0°
7	-0.707	0	0.707	1	45°	180°
8	-0.707	0	-0.707	1	-45°	180°
9	0.707	0.707	0	1	0°	45°
10	0.707	-0.707	0	1	0°	-45°
11	-0.707	0.707	0	1	0°	135°
12	-0.707	-0.707	0	1	0°	-135°

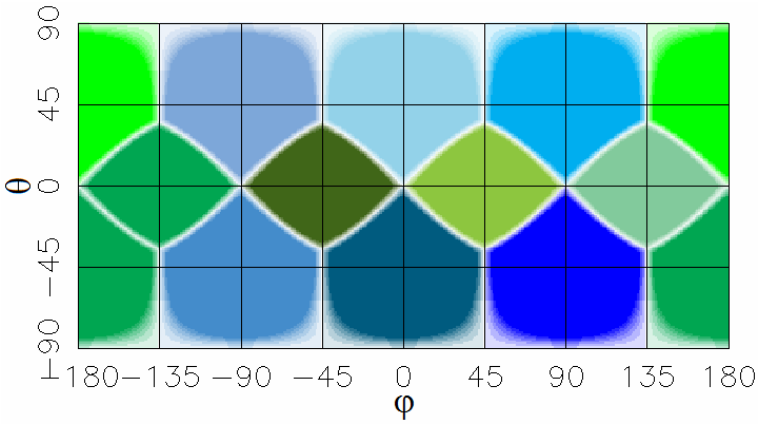


Figure 3.5: Orthorhombic palette

Palette which is created from all of previous palettes has 26 of spontaneous states. Coordinates of all spontaneous points in space has been already defined in all three previous tables. The result palette is shown in Figure 3.6.

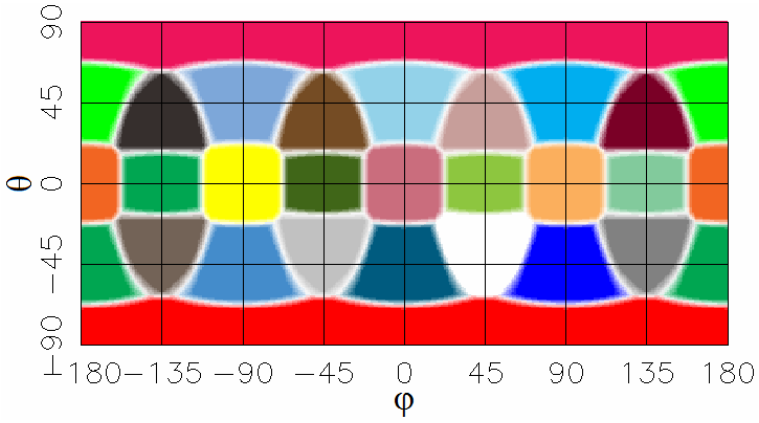


Figure 3.6: Tetragonal + Orthorhombic + Rhombohedral palette

The last palette (Figure 3.7) does not really belong between basic palettes, but I mention it here, because it's just a little modification of tetragonal palette. The only difference is that opposite vectors has the same full color.

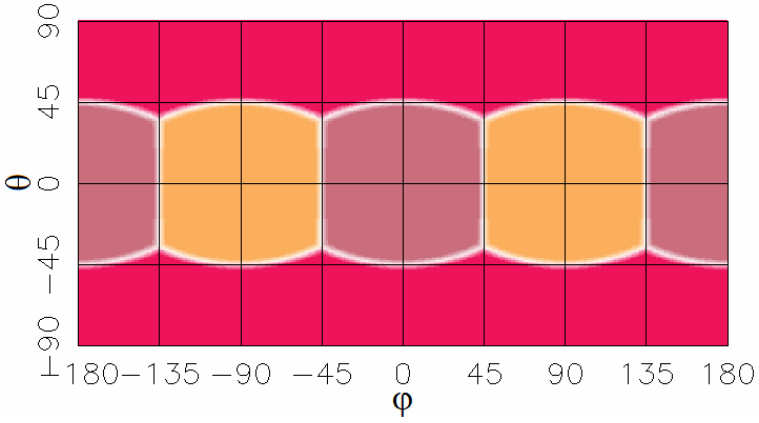


Figure 3.7: Tetragonal palette without different between opposite vectors

3.3 In-plane palette with no significant direction

This special palette was created according to the HSL color model. There are three parts which affect the resulting color. We can imagine the whole HSL color model like a cylinder (Figure 3.8). Hue is angle around the cylinder, saturation value determines the position from the center of cylinder to the edge and lightness gives position from the bottom to the top of the cylinder. At the bottom of the cylinder is black color and on the top is white color.

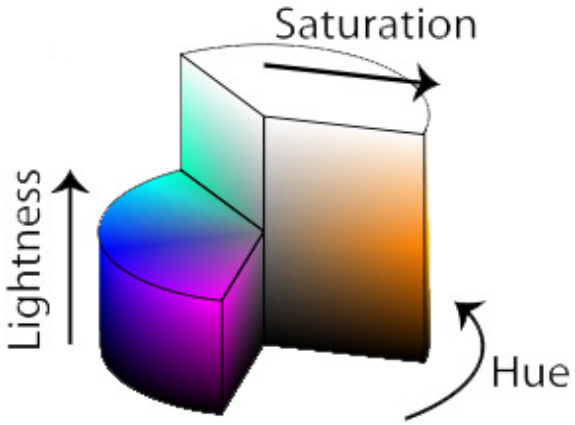


Figure 3.8: HSL Color model (Reprinted from [18])

Two palettes created from HSL color model can be used in `fview`. Palette shown in Figure 3.9 depends only on changing hue parameter. Parameter lightness is set to value 0,5 and saturation is maximal. The number of spontaneous states is equal 360. For every angle of hue is the color different.

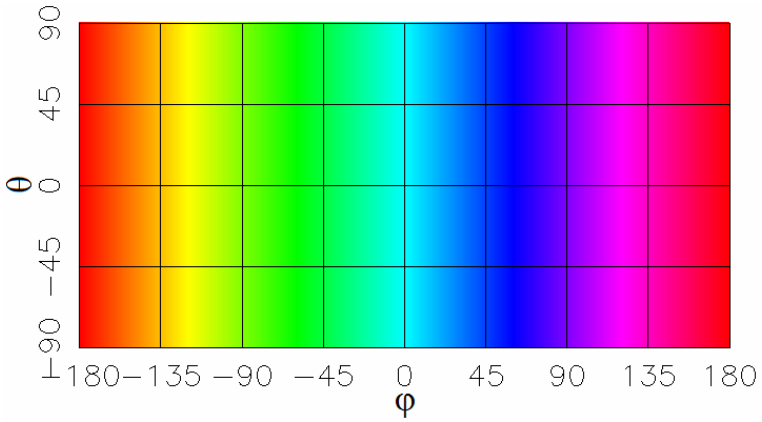


Figure 3.9: HSL color palette with L = 0.5

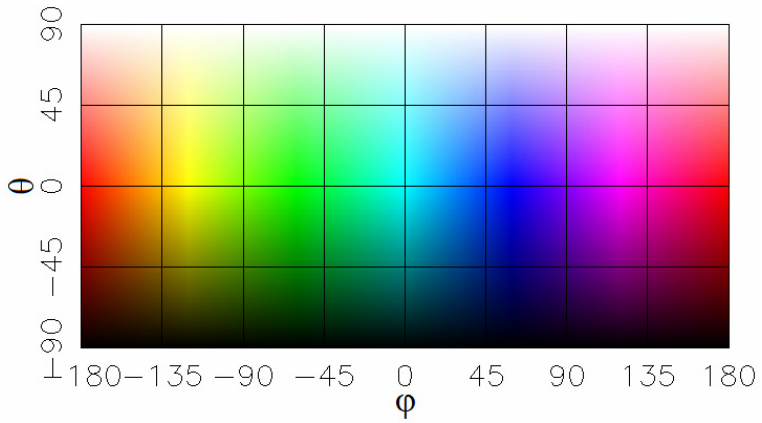


Figure 3.10: Full HSL color palette

Second case (Figure 3.10) differs from the first case that value of lightness is also

changing, saturation stays at maximal value. In this case the number of spontaneous states counts 64 800.

3.4 Grayscale palettes

The last part regarding palettes includes three grayscale palettes using different mathematical dependencies which can be used and plot. They are based on tetragonal palette. Figure 3.11 shows linear growing intensity using equation (5). In all following three examples is range $t \in \langle -1,1 \rangle$. The wider range is the more surface will be covered by shades of black.

$$y = -|t \cdot 255| + 255 \tag{5}$$

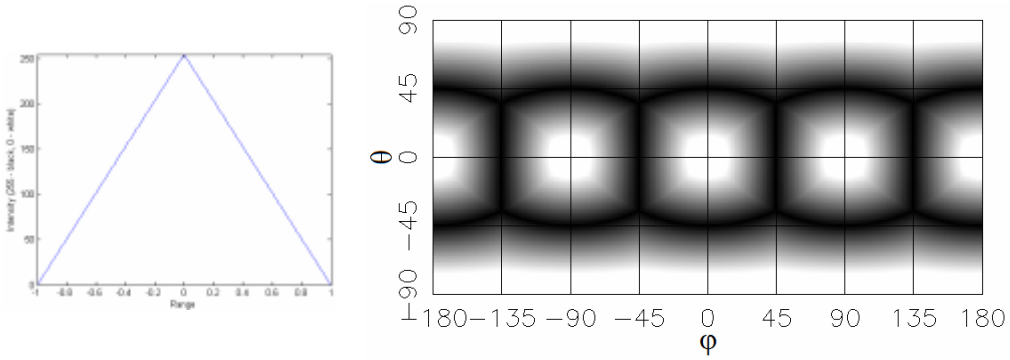


Figure 3.11: Linear function and corresponding palette

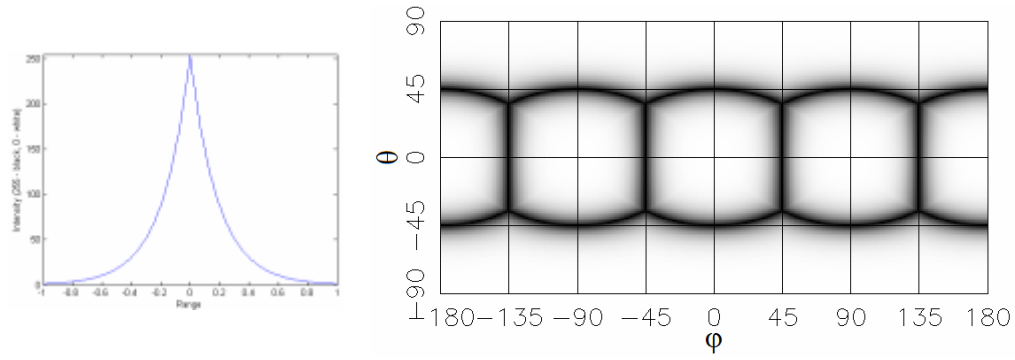


Figure 3.12: Exponential function and corresponding palette

Second palette is nonlinear with an exponential dependence (Figure 3.12). The value 5,3 in exponent is chosen purposely in order to make function grow from -1 (6). The higher is this value, the narrower graph of function will be.

$$y = e^{-|5,3 \cdot t|} \cdot 255 \tag{6}$$

The last case has also exponential dependence with added another parabolic dependence (7). The resulting function and palette is shown on Figure 3.13.

$$y = e^{-5,3 \cdot t^2} \cdot 255 \tag{7}$$

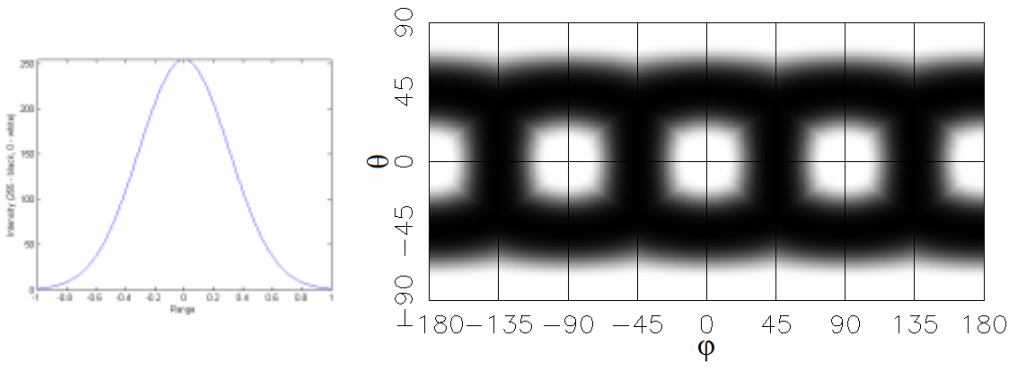


Figure 3.13: Different exponential function and corresponding palette

4 Domain coloring

Every palette which was described in previous chapter can be used for domain coloring. Domain coloring is the process when input data from input file are taken and for each point (composed of one, two or three dimensional vector of polarization in Cartesian coordinates) is computed angles θ and φ. Then every point gets color based on angle values and actual palette.

4.1 Borders

This function has been already implemented in former `fview`. The task is to highlight borders among domains. Algorithm is based on searching neighboring points. It looks around each point in the image and if the spontaneous state is changing then thin black line is drawn. The disadvantage of this method is discontinuity of a line or double border can be found. Figure 4.1 shows subset of images, the left one is without borders and the right is with borders.

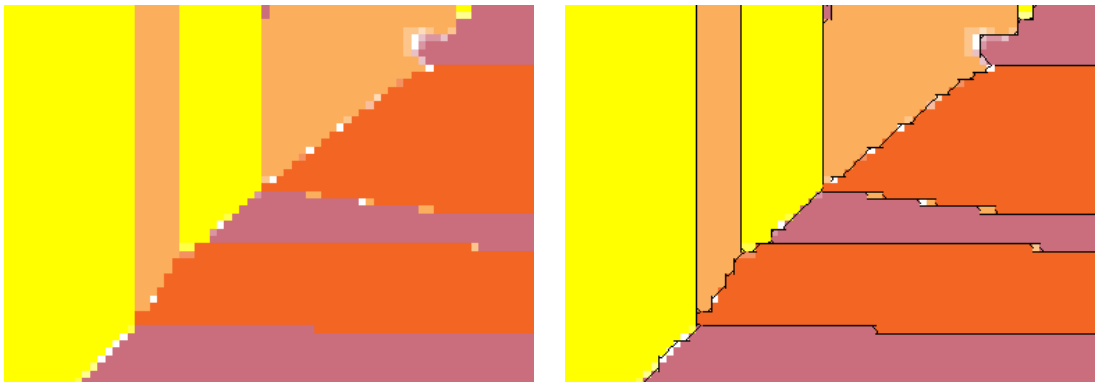


Figure 4.1: Without and with borders

4.2 Small arrows

Next function has been also implemented in former `fview` and is used to display direction of polarization of domains with the arrows. In the Figure 4.2 is an example where domains are colored using tetragonal + rhombohedral + orthorhombic palette and small arrows shows the direction of polarization. The disadvantage is that arrows are not so good visible, especially printed on the paper or if the picture is projected. From this reason I was confronted with the task to display direction of polarization using one big arrow for each domain.

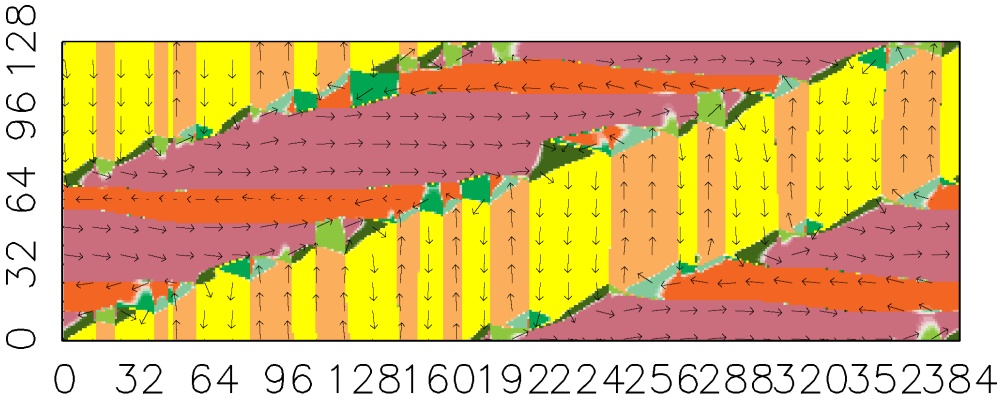


Figure 4.2: Small arrows in practical example

4.3 Flood fill algorithms

Ferroelectric domains are characterized by the same color assigned by the chosen palette. Thus, the task reduces to search for areas with the same attributed color, which nevertheless sometimes have rather complicated shape. This can be accomplished using an algorithm known as flood fill, which is very popular, and is used in many paint programs (well known like bucket tools) to fill bounded area with a given color. Sometimes it is referred to as 'Seed fill'. The first (starting) seed is chosen and the algorithm searches for points with the same color as the first one, and replace it by a new color, until the area is completely filled. There are several implementation of the algorithm. In the following, I mention those relevant for this master thesis.

4.3.1 '4-way flood fill' algorithm

This variant of flood fill algorithm is basically realized by recursive function. This is a special function which can call itself many times. It's very useful when we investigate unknown area. Disadvantage of any recursive algorithm is possible error caused by stack overflow. This situation can occur when the function is called a lot of times and recursion becomes deeper and deeper.

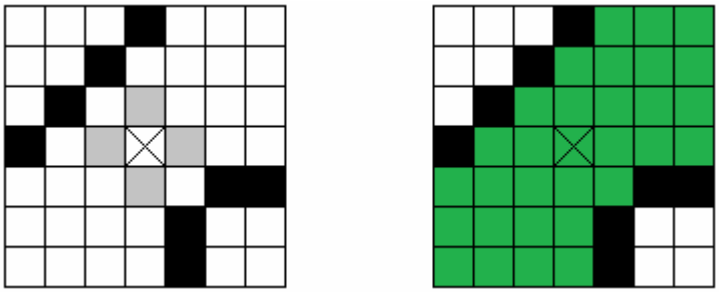


Figure 4.3: 4-way flood fill

Left picture on Figure 4.3 shows initial state before flood fill procedure. The cross indicates the starting point (the first seed). Starting point can be wherever, in this case is in the center of the picture. Grey fields denote basic four directions (west, east, north and south) which can be investigate by this function. Target color is white and replacement color is green. Firstly, pixel with cross is colored in green. Then one of grey directions is chosen (depends on particular implementation) and exactly the same function is called, just with different coordinates, let's say one step to the west. This is repeated until it reaches pixel with different color or the end of the image. If this situation occurs, it tries another direction. As we can see from the right picture, pixels that are connected on diagonal line were not found.

4.3.2 '8-way flood fill' algorithm

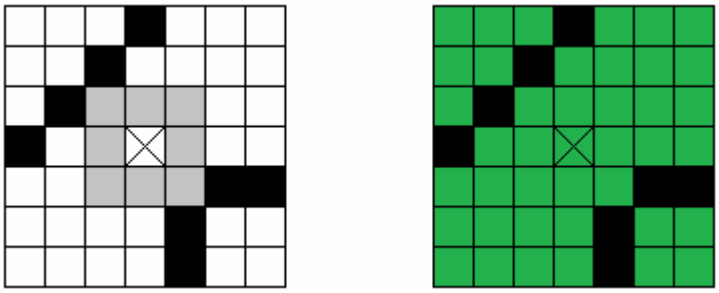


Figure 4.4: 8-way flood fill

The same picture, the same starting point, but 8-way recursive flood fill algorithm (Figure 4.4). The difference is obvious. Algorithm can move to every direction around the point. It means it will find all points, even if they are connected only in diagonal direction. If we have object bordered just by 1 pixel thin line and we want to fill it inside, 8-way algorithm will leak through border outside of the object. From this reason is not suitable to use it. On the other side this property is using for finding edges of objects as we can see on Figure 4.5.

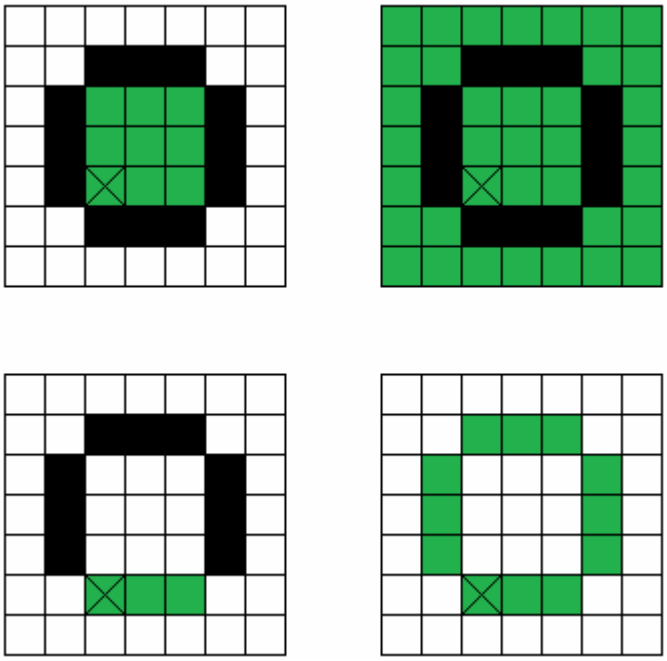


Figure 4.5: Difference between 4-way and 8-way algorithm

Next thing what really influences the performance and the time for processing algorithm is largeness of objects (domains). If the domain is larger than critical limit, recursive algorithm will call itself a lot of times as I mentioned before and the situation called stack overflow occurs. The structures and domains should be small or large and from this reason I needed to implement more sophisticated flood fill algorithm.

4.3.3 'Scanline flood fill' algorithm with stack

The best solution has name scanline flood fill algorithm using my own stack. Basically is the stack LIFO (last in, first out) linear data structure and we are working only with the top of the stack. So we are using well knows operations push (for store data to stack) and pop (to load data from stack). Last operation which is possible to do with stack allows deleting all data stored in the stack.

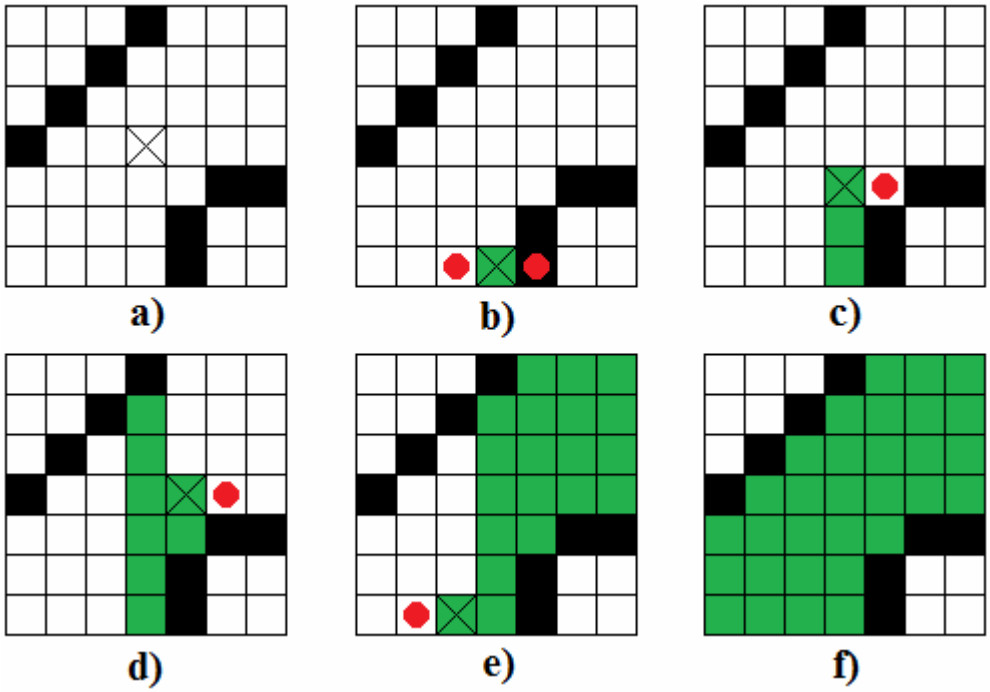


Figure 4.6: Scanline process

Figure 4.6 shows 6 images describing how scanline algorithm works. We want to fill area with given boundaries. Black cross marks the start point (4.6.a). Firstly we need to go down in y-axis and find the lowest point with the same color as start point and color it with a green color. From this point we investigate colors of the left and right neighbors (red fill circles). The left neighbor has the same color like starting point, so we push it to stack and also set a variable which gives the information that left neighbor has been chosen. Right neighbor is boundary, so it has different color. This point will not be pushed into stack (4.6.b). Algorithm continues with filling in vertical direction from bottom to top. When it reaches third pixel from bottom (4.6.c), it detects a right neighbor which fulfill conditions and push it to stack. While it reaches top of the picture or point which has different color, it calls pop function. On the top of the stack is right neighbor, so it starts filling and looking for the same color neighbors again. Whole procedure is repeated until the area is filled.

4.4 Big arrows

Since I was able to find exact area I need to draw big arrow in the direction of polarization inside this area. Size of the arrow is dependent on largeness of the area. Arrows is always drawn to the center of gravity of flood filled area. Figure 4.7 shows an example of domain structure with big arrow for each domain.

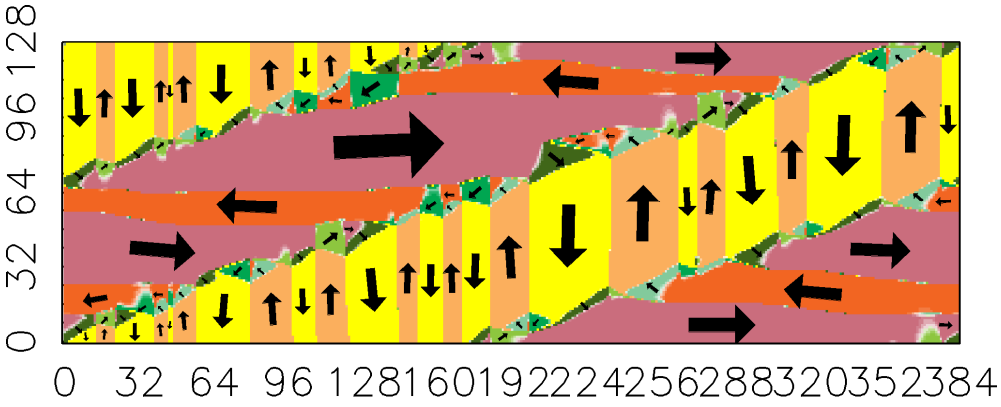


Figure 4.7: Big arrows in real image

4.5 Big arrows and three dimensional polarization

Another advantage of big arrows unlike the small arrows is that images with three dimensional ferroelectric polarization can make an impress of perspective view. If the angle between z-axis and plane (x and y-axis) is less than 60° (it means z component of polarization is positive) then arrow has bigger head. In other case, if angle is greater than 120° then arrow has bigger tail (Figure 4.8).

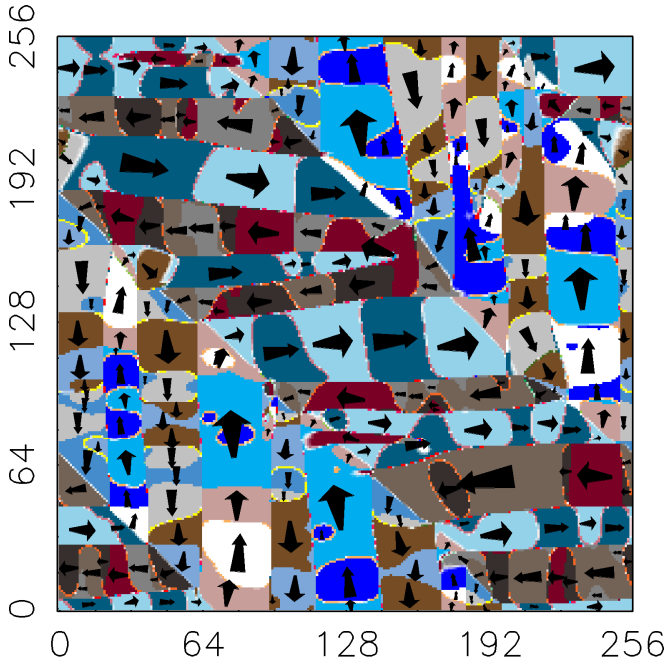


Figure 4.8: Big arrows and z component of polarization dependency

The last special case occurs when three dimensional polarization contains just the last z component (other ones are zero). In this case arrows pointing up or down in the direction of z-axis. If the arrow pointing down, is it illustrated as circle with cross

inside, if arrow pointing up, then it is circle with dot inside (Figure 4.9). On this picture is also significant disadvantage of method which draws arrows in the center of gravity. This problem is regarding for cases with one or two dimensional polarization as well. It is also harder to determine which arrow corresponding with appropriate domain. The worst case is complex structure with holes inside. In this case with high probability will arrow interfere with another domain. The best way how to solve this issue could be algorithm which should adapt size and position of arrows.

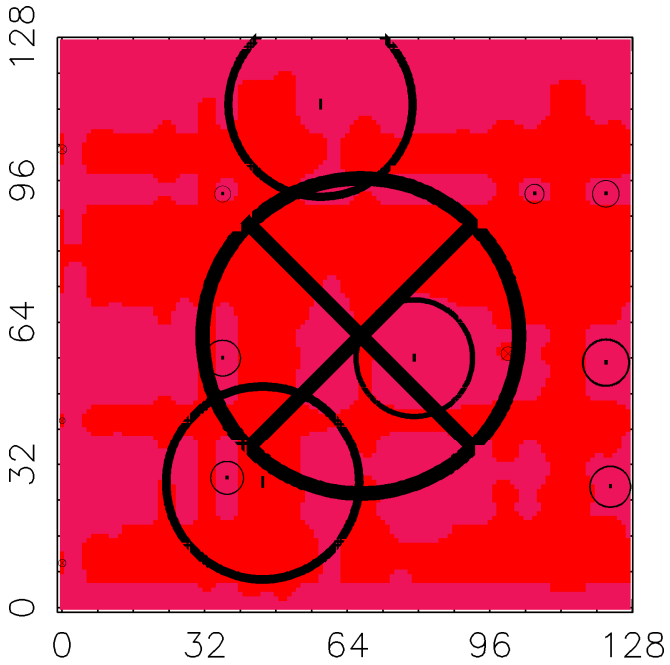


Figure 4.9: Big arrows represented by circles

4.6 Defects

The last part related to domain coloring is defects in ferroelectric materials. They are always presented and it is inseparable part which is needed to deal with in realistic simulations. To illustrate defects in the image is necessary to load two files. One file is classical input file (Figure 4.10.a). Second file (using parameter *–t*) contains just ones and zeros. Width and height of original file and defect file has to be equal. Everywhere the defect file contains ones there small black square is drawn to the original image (Figure 4.10.b).

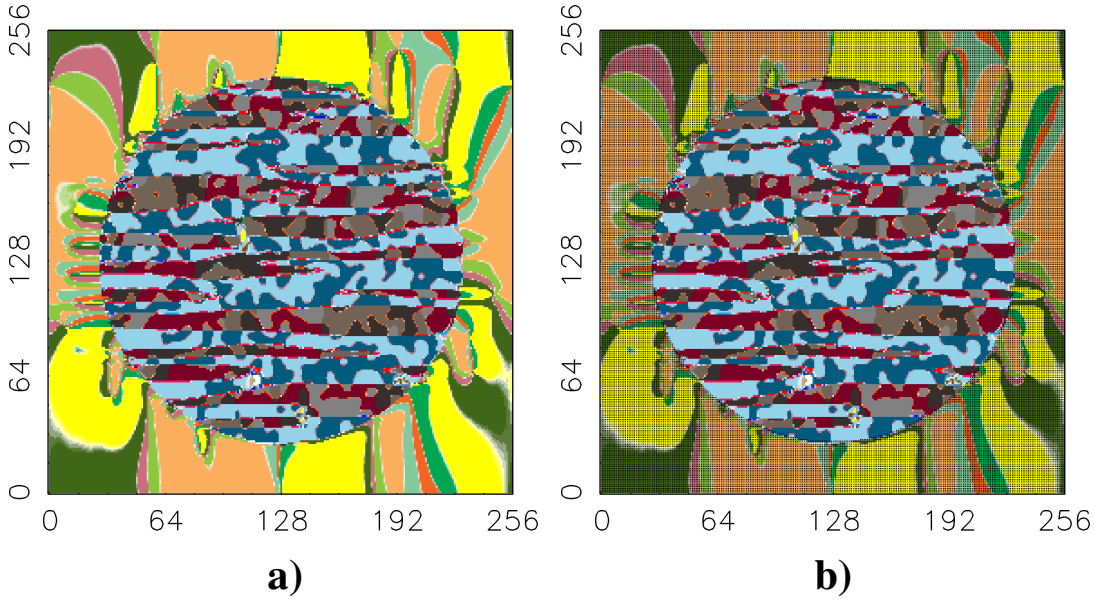


Figure 4.10: Defects representation

5 Sharp interface among domains

Another task in my diploma thesis was to implement function to program `fview` for finding sharp interface among ferroelectric domains. Basic idea is to find exact points representing interface and consequently interpolate these points by curve using Ferguson's cubic.

5.1 Ferguson's cubic

Ferguson's cubic [19] are third order polynomial curves, which are widely used for interpolation points. Ferguson's cubic is determined by starting point P_1 (P_{1x}, P_{1y}) and end point P_2 (P_{2x}, P_{2y}) and by two vectors $\overrightarrow{p_1}(p_{1x}, p_{1y})$ and $\overrightarrow{p_2}(p_{2x}, p_{2y})$ in these points in two dimensional space. The resulting curve depends on size and direction of vectors. Parametrical equation follows.

$$P(t) = P_1 F_1(t) + P_2 F_2(t) + p_1 F_3(t) + p_2 F_4(t) \tag{8}$$

Where F_1, F_2, F_3 and F_4 are Hermit polynomials and $t \in < 0,1 >$

$$\begin{aligned} F_1(t) &= 2t^3 - 3t^2 + 1 \\ F_2(t) &= -2t^3 + 3t^2 \\ F_3(t) &= t^3 - 2t^2 + t \\ F_4(t) &= t^3 - t^2 \end{aligned} \tag{9}$$

For connecting curves it's necessary to fulfill equality of point P_1 of following curve with point P_2 of actual curve. Smoothness is guaranteed if vector $\overrightarrow{p_2}$ of actual curve is identical with vector $\overrightarrow{p_1}$ following curve.

5.2 First step for finding interface

First of all it is necessary to mention that flood fill algorithm is used again for this functionality. Figure 5.1 shows three types of situations that can occur between two different domains (red and green). White color represents interface. First image (5.1.a) is the easiest situation which occurs very rarely. Two different domains has only 1 pixel

thin interface between. Image 5.1.b contains two domains with different polarization of vectors, but full color is followed by another full color so there is not any interface. Last and very common situation is shown on the image (5.1.c) where full color gradually changes to white color. In the beginning before I make first pass of flood fill, I create a domain array of the same size as the picture and initialize it to zero values. With every call of flood fill is domain counter increased and this value is used for whole area in domain array. Let's consider number 1 representing red color in domain array and logically number 2 representing green color (Figures 5.1.d, 5.1.e and 5.1.f). Because flood fill do its job only with full colors, domain array contains zero values where lighter shades of red and green colors are presented (5.1.c, 5.1.f). In this case is an interface 5 pixel wide.

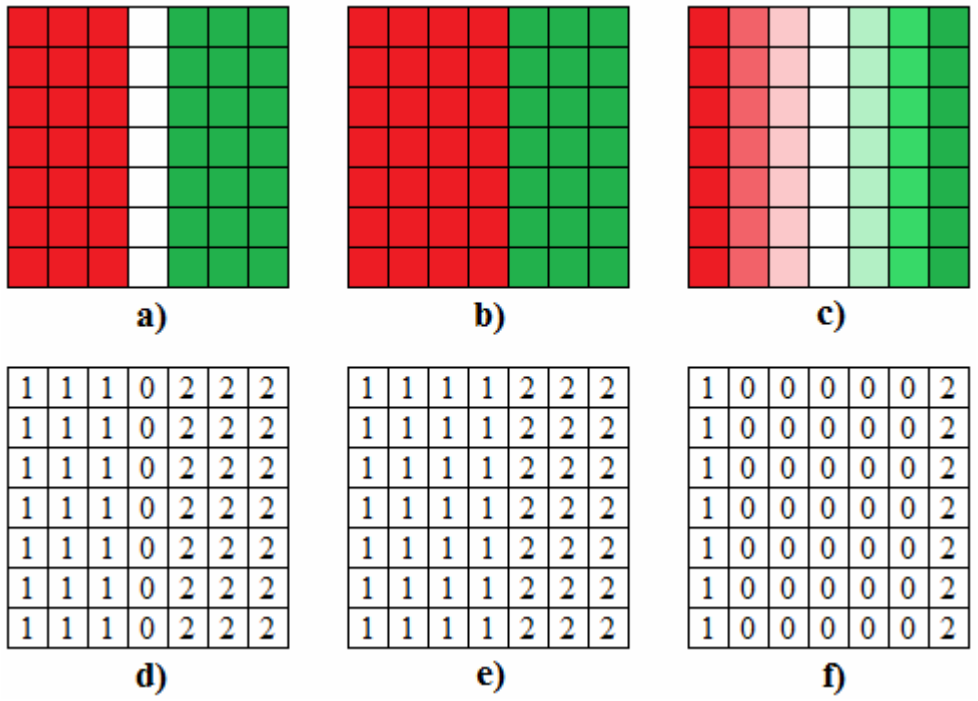


Figure 5.1: Three possibilities and corresponding domain arrays

In the first case (Figure 5.1.a) is possible to interpolate all found points without any further modifications. Second case (Figure 5.1.b) need to be modified so, that after the flood fill algorithm is completely done for whole picture, it is necessary to find areas where domain number is changing to another and artificially add zero values. Result will be as in Figure 5.2.

1	1	1	1	2	2	2
1	1	1	1	2	2	2
1	1	1	1	2	2	2
1	1	1	1	2	2	2
1	1	1	1	2	2	2
1	1	1	1	2	2	2
1	1	1	1	2	2	2

1	1	1	0	2	2	2
1	1	1	0	2	2	2
1	1	1	0	2	2	2
1	1	1	0	2	2	2
1	1	1	0	2	2	2
1	1	1	0	2	2	2
1	1	1	0	2	2	2

Figure 5.2: Artificially added zeros

Unfortunately there is a third case (Figure 5.1.c) which makes thing more complicated. It's impossible to interpolate all zero values. The task is to find accurate position between domains even if the interface is wider.

5.3 Search for accurate position of interface between domains

To fulfill this task I implemented gradient function in 3D, which is consequently used to evaluate the biggest gradient (10). Because images loaded by `fview` can have one, two or three dimensional polarization I needed to choose the biggest partial derivative using (11).

$$\nabla f(x,y,z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) \tag{10}$$

$$\frac{\partial f}{\partial x} = f(x+1,y,z) - f(x-1,y,z)$$

$$\frac{\partial f}{\partial y} = f(x,y+1,z) - f(x,y-1,z)$$

$$\frac{\partial f}{\partial z} = f(x,y,z+1) - f(x,y,z-1) \tag{11}$$

There is one special case which I didn't mention and which I solved together with finding biggest gradient. The problem concerns the case when interface is between two domains which have the same color (Figure 5.3). These zero points should not be taken into account because it's unwanted to draw interface between domains in this case. Algorithm will be described in following sentences.

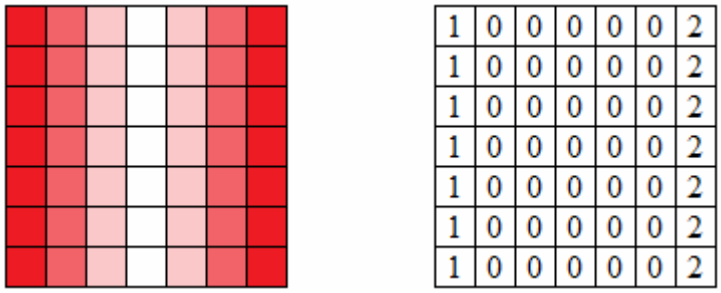


Figure 5.3: Interface between same colors

At this time algorithm goes through whole domain array in order to find zero value. Figure 5.4.a shows example of domain array. When the first zero (marked by red circle) value is found, gradient is computed. Gradient for this point has some direction, which is indicated by blue line (Figure 5.4.b). Algorithm goes firstly from red circle let's say to the left, until it finds different number than zero in domain array (number 1 at this point) or the maximum distance given by `max_g_search`. Then it stores the color of domain and continues with finding gradient on the other direction until it finds number 2 (or also the same distance `max_g_search`) in domain array. If color of domain at the point 2 is equal to color domain at the point 1, then this point will not be taken into account. Here it is necessary to realize that algorithm doesn't compare number in domain array, but real colors.

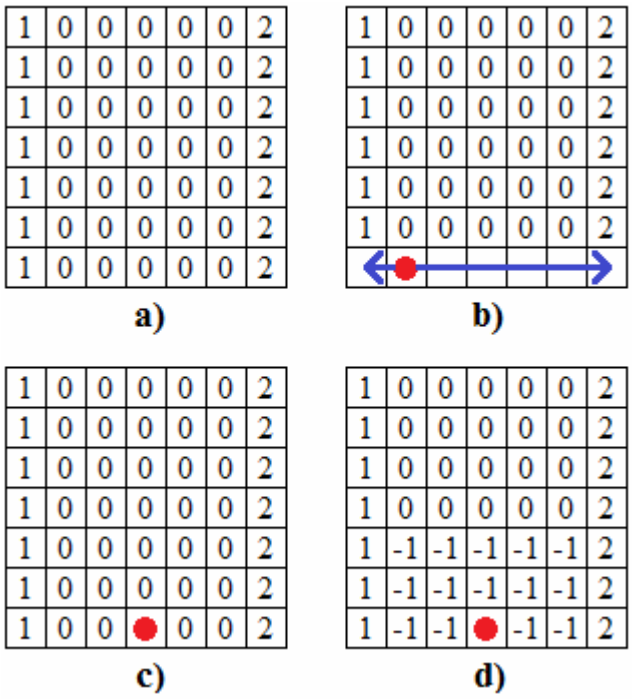


Figure 5.4: Finding the best position between domains

Figure 5.4.c shows point with the biggest gradient. Last step is to mark points in some radius around the origin (Figure 5.4.d) and store this point into array. This is necessary step to obtain good interpolation.

5.4 Sorting points for drawing

Since I have all points on correct position it is still impossible to take all these points and interpolate them by Ferguson's cubic. Because the image is scanned from bottom to top and from left to right, found interface points are not correctly sorted. From this reason I take just those points which separate just two exactly the same domains. This could be sufficient, but in some situation can occur what is shown on Figure 5.5. Red interface points are numbered as they are stored in array and after interpolation we get wrong result (5.5.a) while we expected different result (5.5.b).

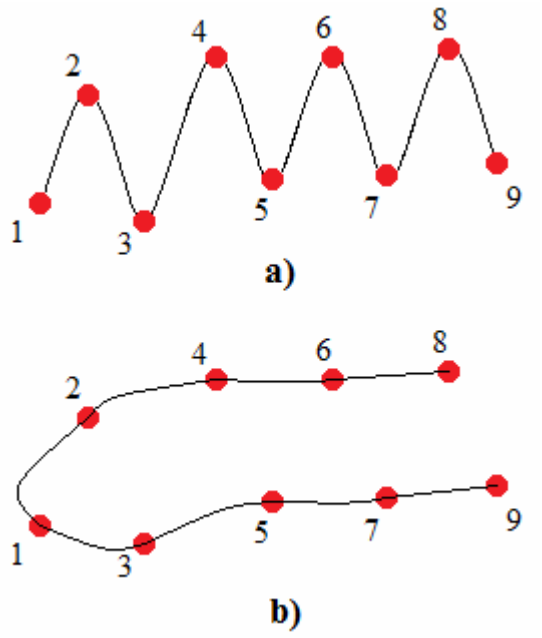


Figure 5.5: Wrong and correct interpolation

In order to solve this kind of situations I decided to look for neighbors in given radius of every single point. Basic idea supposes that point which will have just one neighbor is the end point of curve. From this end point algorithm goes to neighbor, from this neighbor to his neighbor, but no back, until it reaches second end of curve. This would work properly assuming that every point will have one or maximal two neighbors. But if points are closer to each other, one point can have up to four neighbors.

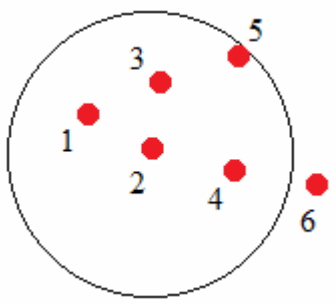


Figure 5.6: Arranged points and neighbors

Figure 5.6 illustrates a situation with more than two neighbors for one point. Big black circle marks area where neighbors are searched for point with number 2 (logically in the center of the circle). So we can see than point 2 has four neighbors. All points and their neighbors are in table 3.

Table 3: Points and their neighbors

Point	Neighbors
1	2,3
2	1,3,4,5
3	1,2,4,5
4	2,3,5,6
5	2,3,4
6	4

The task is to choose the right neighbors. The problem can be solved by finding the maximum direct angle between vectors using (12). The range of angle is (0°, 180°). Figure 5.7 shows an angle between vectors created by connecting points 5-2 and 4-2. For all combinations are angle computed and the best neighbors are stored. For point 2 are the best neighbor points 1 and 4.

$$\cos\alpha = \frac{\left|u_1 \cdot v_1 + u_2 \cdot v_2\right|}{\sqrt{u_1^2 + u_2^2} \cdot \sqrt{v_1^2 + v_2^2}} \tag{12}$$

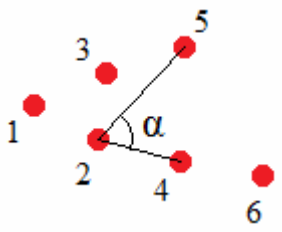


Figure 5.7: Angle between two vectors

Correctness of finding endpoints of curves depends on the radius where we look for neighbors. Figure 5.8 shows situation when the radius is too big and for the first point the neighbors are points numbered 2 and 3. Curve (in this case line) between these points will not be drawn at all, because there is no point with just one neighbor (endpoint). From this reason I implemented a function which checks coordinates of the neighbors. If the neighbors are on the same side (valid for x and y-axis - in this case both points are on the right side to the origin) then artificial endpoint is created.

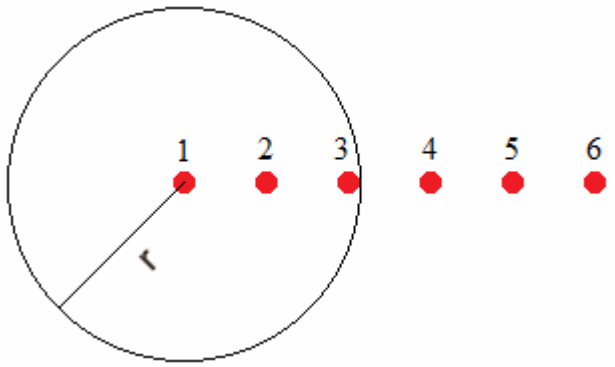


Figure 5.8: Bigger radius and no endpoint

5.5 Connecting rest points

At this time, points which separate two same domains are interpolated by curve in the whole picture. In ideal case every curve has two endpoints. The task is to connect endpoints together in order to make interface continuous. Figure 5.9.a shows subset of the image using tetragonal palette with different domains. Points with the same color separate two same domains. They are placed where is the maximal gradient. As we can see, they are already connected by black curve (in this case it's line). The color of points was randomly computed based on domain numbers which they separate. So in the picture there are points with the same color separating different domains but that's not essential. Figure 5.9.b shows the result of connecting endpoints.

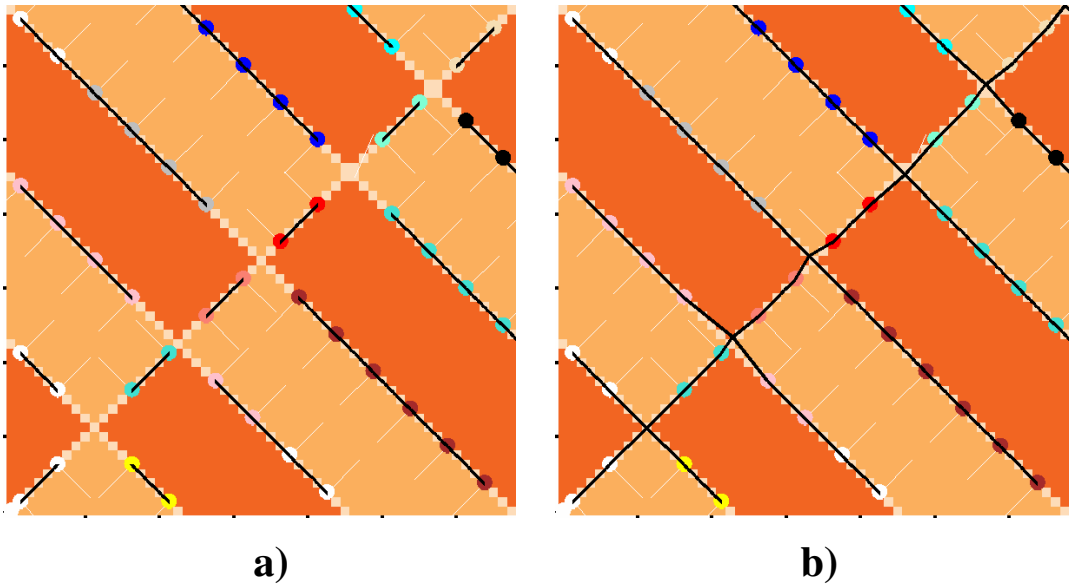


Figure 5.9: Before and after connecting endpoints

Algorithm works so, that if it finds endpoint which was not used, it looks around some radius for another endpoint which has a common domain. When some point is found algorithm looks for further endpoint around this new point. This procedure continues until all points are found. If it founds just two points, they are connected simply by line. In other case if it founds more than 2 points, it computes center of gravity of these all points and subsequently from all points straight line is drawn to this center of gravity.

When the radius defined by `radius_endpoints_search` variable for searching is inappropriately chosen it can found unnecessarily large number of endpoints. From this reason number of points that can be found is limited by `max_endpoints_number` variable. But the reliability of this method is still dependent on the size of radius. Figure 5.10 shows the same image with smaller radius for searching endpoints. The result is now apparently worse. This problem is not very easy to solve using this method. One of the possibilities should be set bigger radius and take just the closest endpoint to the origin endpoint.

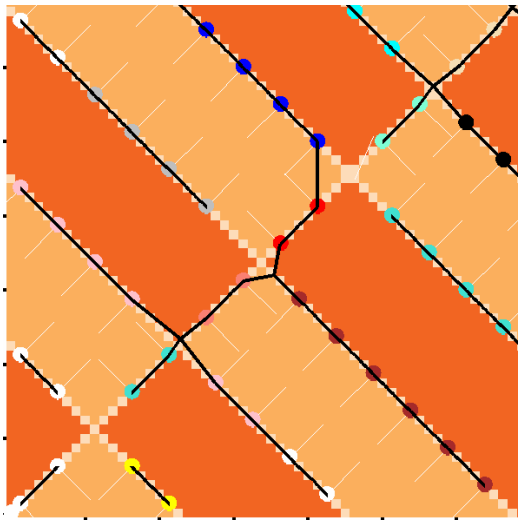


Figure 5.10: Smaller radius for finding endpoints

At the end of this chapter I compare borders with sharp interface on the same subset of image used in chapter 4 (Figure 4.1). More examples are in appendix.

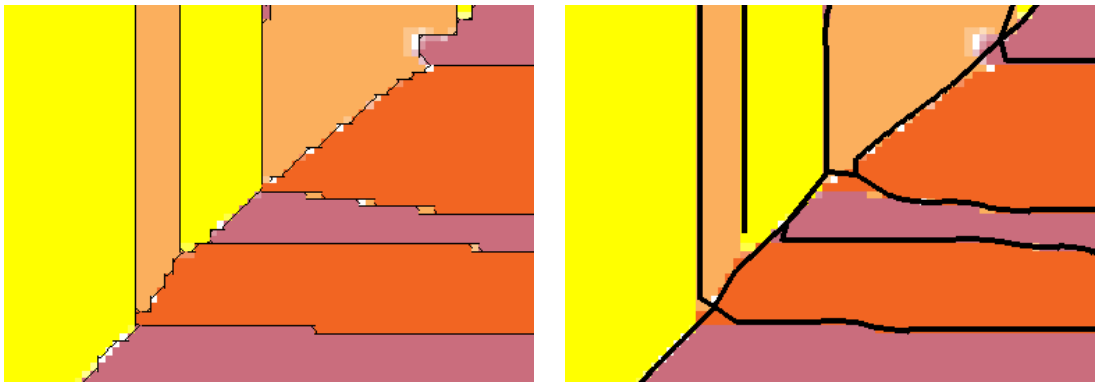


Figure 5.11: Comparison between borders and sharp interface

Conclusion

The main outcome of this work is implementation of new functionalities into program `fview`. Also, whole program has been rewritten using third-party cross-platform PLplot library for visualization, because PGPLOT (old library for visualization) is no longer supported. First and important new function is ability to draw one big arrow for each domain, which size is dependent on largeness of the domain. For identifying sometimes very complicated structures of domains is used flood fill algorithm. The arrows can also create the impression of perspective view when z-component of polarization fulfill given conditions. It is supposed to facilitate the interpretation of results. The main disadvantage is that in some cases arrow can interfere to other domain. Second important function is searching for interface among domains using interpolation by Ferguson's cubic. Points which belong to the interface are identified by using the flood fill algorithm again or they are artificially created. Correct position of the points is provided by maximal gradient computing. The proper order of connection points is ensured by the most direct angle between two vectors, which is not the best solution every time. Generally it works for most cases, but it is very sensitive to the parameter settings.

The program is able to process commands from the command line. Data can be visualized in several specified regimes; user can choose and combine many palettes with many options. `Fview` is now able to plot and use basic palettes (tetragonal, orthorhombic, rhombohedral and tetragonal + orthorhombic + rhombohedral) and newly HSL color model in two modes and three types of grayscale palettes based on tetragonal palette.

At the end I can tell that requirements of this diploma thesis have been successfully accomplished. Shortcomings of my solutions and suggestions for improvements are mentioned. I hope this diploma thesis will be useful for future development of program `fview`.

References

[1] Ondrejkovič, P. *Studium doménových struktur ve feroelektrickém BaTiO3*, Master thesis, ČVUT (2008)

[2] Pulpan, P., Erhart, J. *Piezoelektrické chytré materiály pro elektrotechniku*, Elektro 2002/11, 4 (2002)

[3] Herout, P. *Učebnice jazyka C 1. a 2. díl,* KOPP (2004)

[4] Source code of the program `fview` for the visualization of the domain structure; Documentation of the software package Ferrodo

[5] Lackner, R. *RLPlot* [online]. 2012-04-28 URL: <http://rlplot.sourceforge.net/>

[6] *OpenDX* [online]. 2012-04-28 URL: <http://www.opendx.org/index2.php>

[7] Franke, K. *SciDAVis* [online]. 2012-04-28 URL: <http://scidavis.sourceforge.net/>

[8] Grubb, S. *ploticus* [online]. 2012-04-28 URL: <http://ploticus.sourceforge.net/doc/welcome.html>

[9] *EasyViz* [online]. 2012-04-28 URL: <http://code.google.com/p/scitools/>

[10] Vasilief, I. *QtiPlot* [online]. 2012-04-28 URL: <http://soft.proindependent.com/qtiplot.html>

[11] *GLE* [online]. 2012-04-28 URL: <http://glx.sourceforge.net/>

[12] Hester, D. *PLplot* [online]. 2012-04-28 URL: <http://plplot.sourceforge.net/>

[13] Hunter, J., Dale D., Droettboom M. *matplotlib* [online]. 2012-04-28 URL: <http://matplotlib.sourceforge.net/>

[14] Williams, T., Kelly C. *Gnuplot* [online]. 2012-04-28 URL: <http://gnuplot.info/>

[15] Eaton, W. John *GNU Octave* [online]. 2012-04-28 URL: <http://www.gnu.org/software/octave/>

[16] Digiteo *Scilab* [online]. 2012-04-28 URL: <http://www.scilab.org/>

[17] Ramachandran, P. *MayaVi2* [online]. 2012-04-28 URL: <http://docs.enthought.com/mayavi/mayavi/index.html>

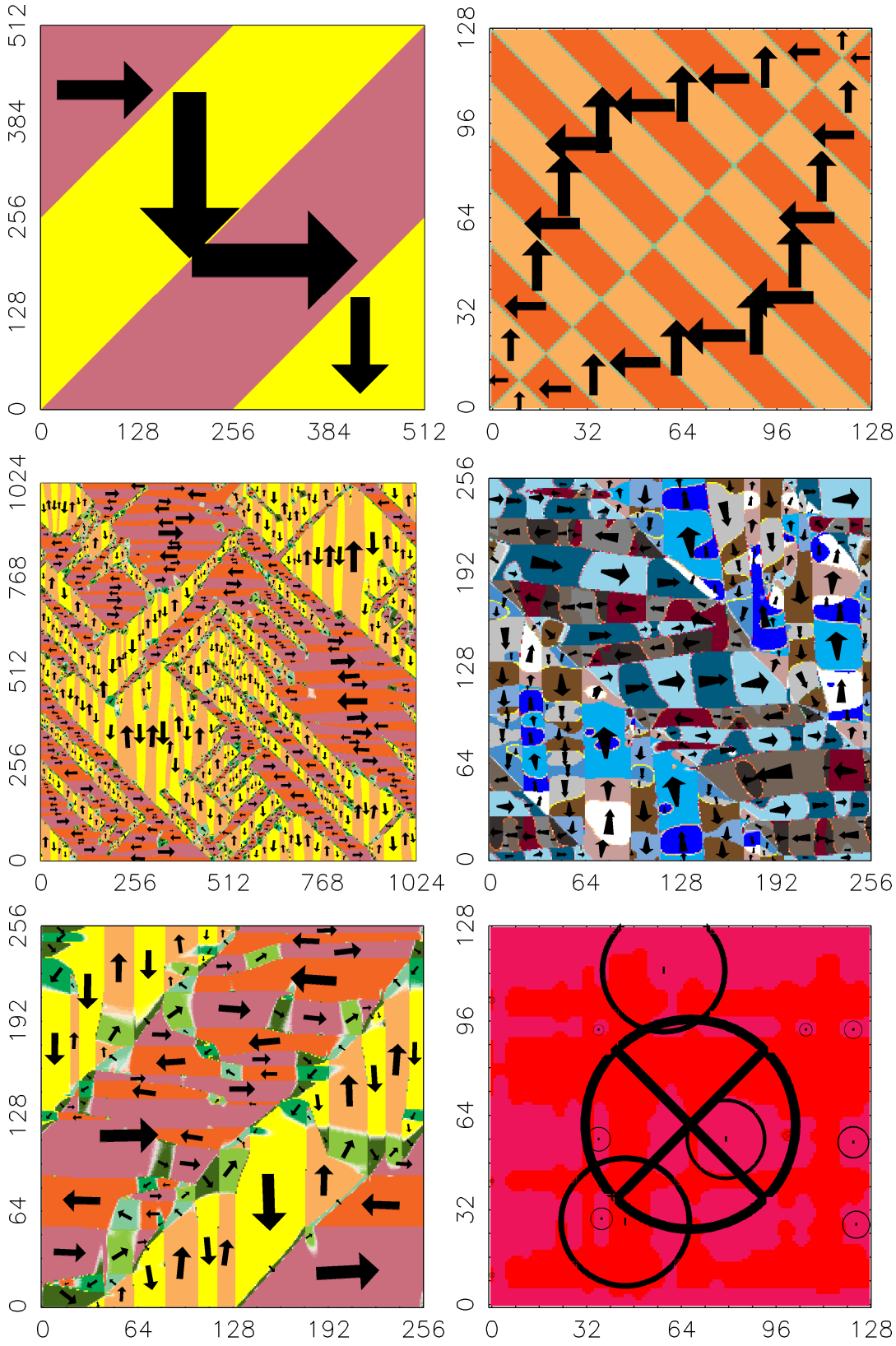
[18] *HSL Color space* [online]. 2012-04-28 URL: <http://en.wikipedia.org>

[19] Žára, J., Beneš, B., Felkel, P.: *Moderní počítačová grafika*, 448 stran, ISBN 80-7226-049-9

Appendix A

Following configurations were plotted using command:

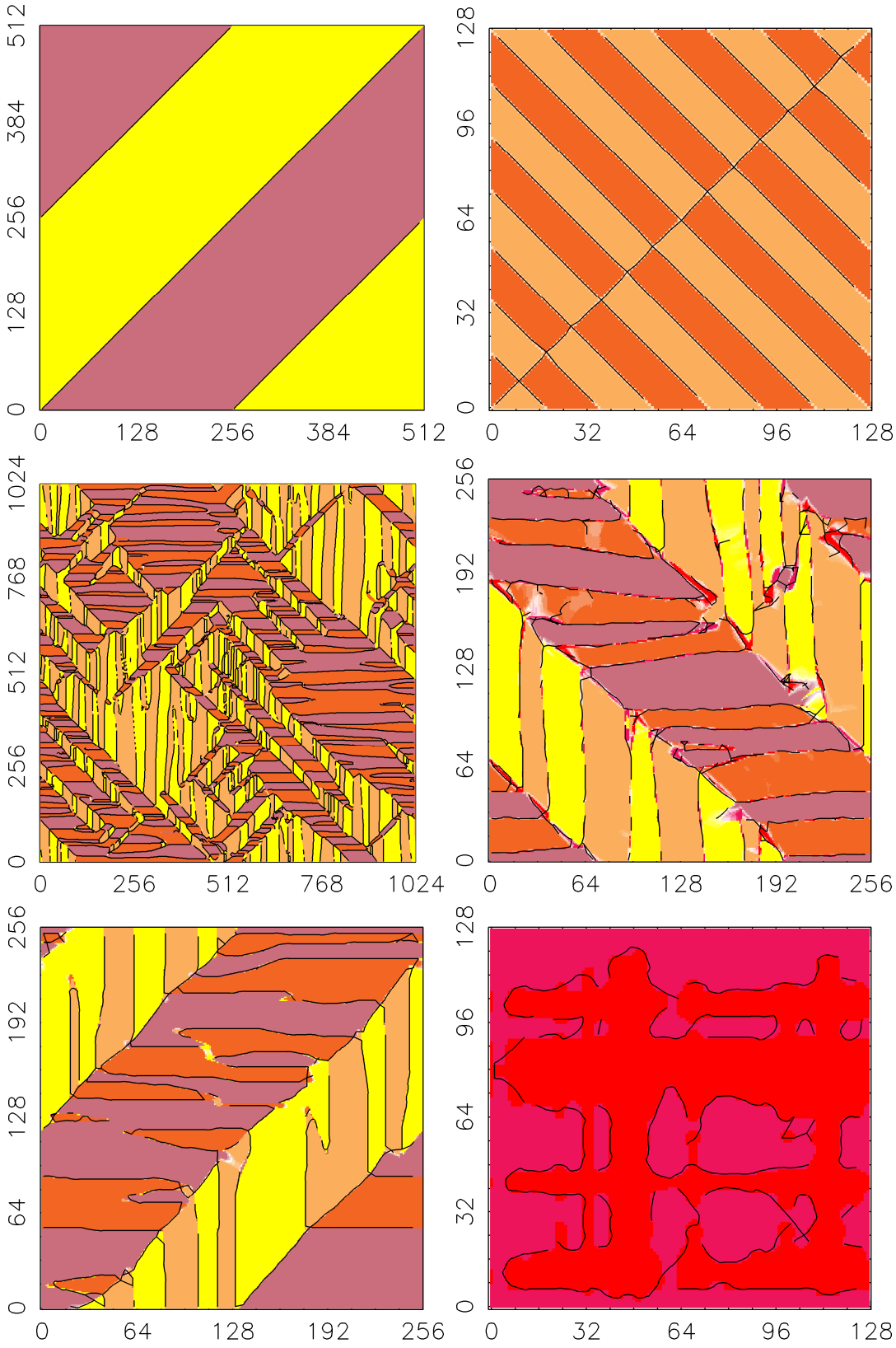
```
fview -p 0 -l color#big_arrows
```



Appendix B

Following configurations were plotted using command:

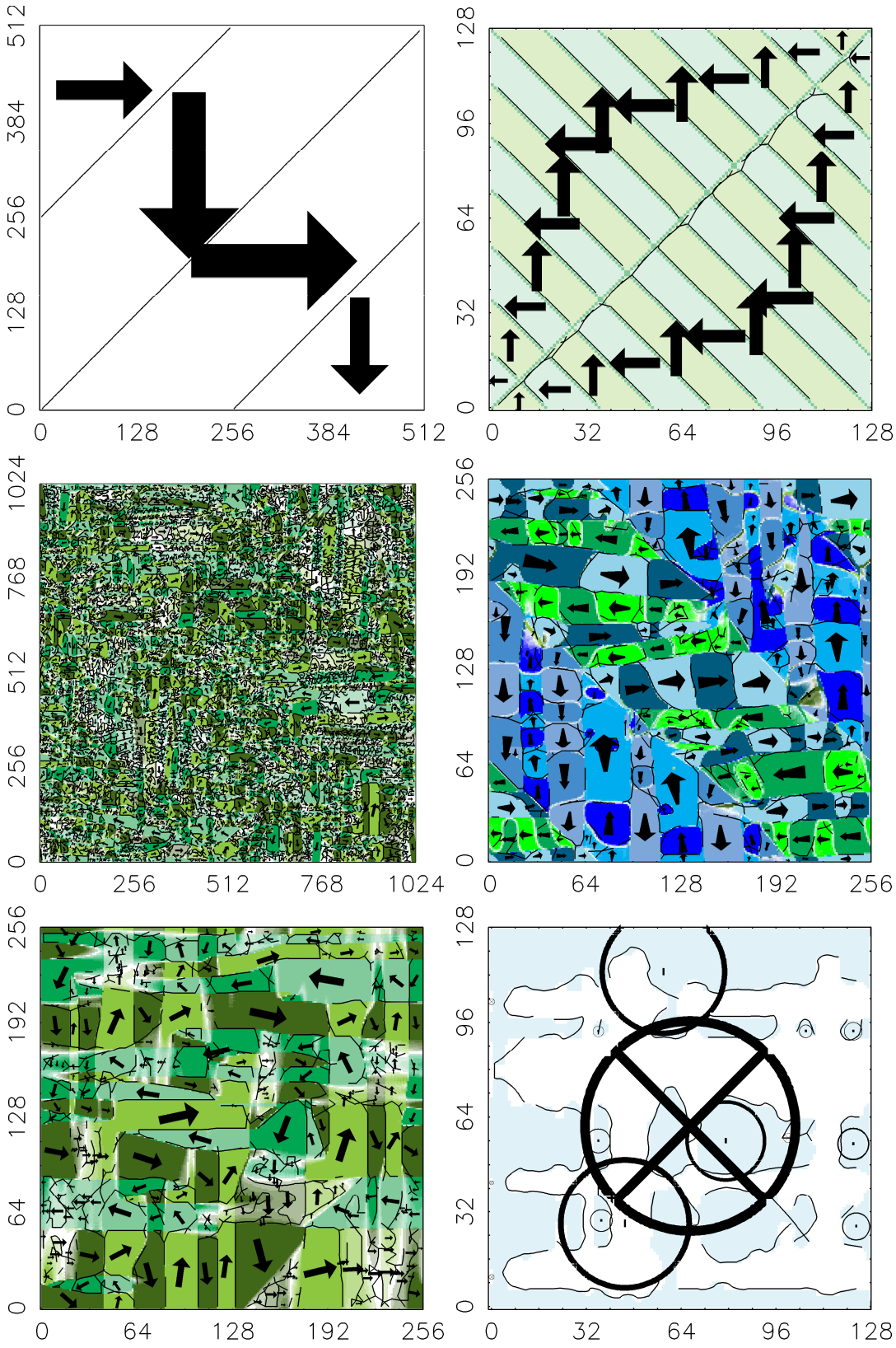
```
fview -p 1 -l color#sharp_interface
```



Appendix C

Following configurations were plotted using command:

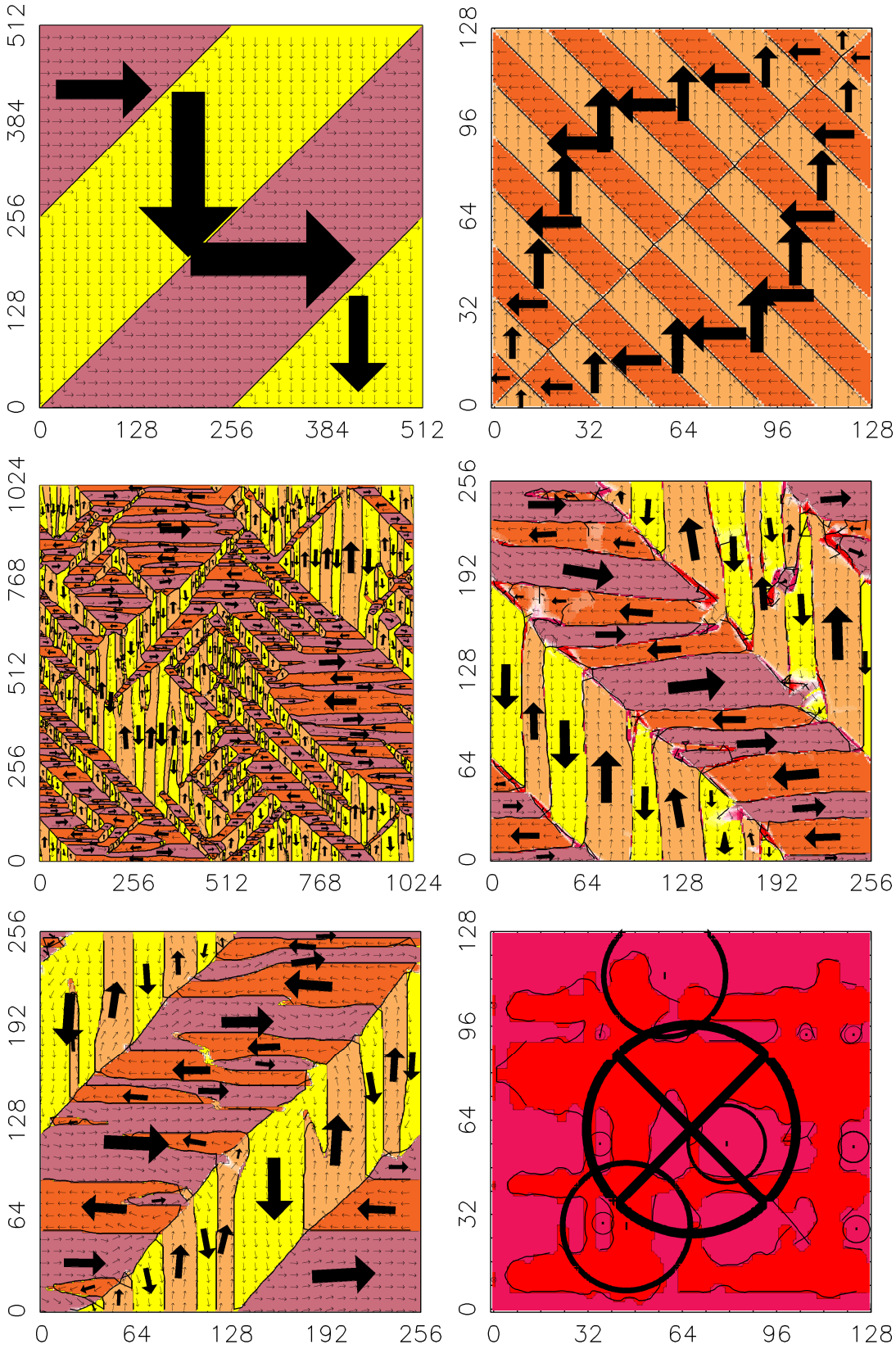
```
fview -p 2 -l color#big_arrows#sharp_interface
```



Appendix D

Following configurations were plotted using command:

```
fview -p 1 -l color#borders#small_arrows#big_arrows#sharp_interface
```



Appendix E

Following configurations were plotted using command:

```
fview -p 9 -l color#small_arrows
```